ABCM Symposium Series in Mechatronics - Vol. 2 - pp.676-682

REAL TIME ACTION PLAN SYNTHESIS AND EXECUTION IN INTELLIGENT MANUFACTURING SYSTEMS

Marcelo Nicoletti Franchin
UNESP – São Paulo State University, Av. Luiz Edmundo Carrijo Coube, s/n, CEP 17033-360 – Bauru - SP – Brazil
e-mail: franchin@feb.unesp.br

Marcio Rillo
USP – University of São Paulo, Politechnic School, Av. Prof. Luciano Gualberto, 158, trav3, CEP 05508-900, São Paulo – SP – Brazil
e-mail: rillo@lsi.usp.br

Abstract. *This work presents an AI/planning method for autonomous intelligent agents, named ALWAYS$_{TRX}$, that handles external events in the planning phase, different from the existing planning methods. An intelligent manufacturing system, a robot or some other form of actuator with a control structure that can execute an action plan, collect information about the environment and use automated reasoning techniques about its perception and about what action to be done, can be called intelligent agent. Planning with external events makes possible reasoning about what action to execute, dealing with eventual world state changes that may occur during a mission. World state changes are obtained by the planner through the sensory system. The ALWAYS$_{TRX}$ method integrates planning with the control system, using an unique knowledge representation, where the plan being generated specify the actions to be taken and the external events to be observed. Among the heuristics used by the ALWAYS$_{TRX}$ method to expand and search in the planning tree, are the weights assigned to the occurrence of the possible external events that may happen in a given world state. With these characteristics, the ALWAYS$_{TRX}$ method is an effort to use planning systems in real world applications, typically complex, dynamic and with real-time restrictions. This method was implemented using C language and an adaptation of RETE match algorithm was designed to increase the planner performance.*

Keywords*: Real Time AI/Planning, Planning and Execution, Planning with events, Mechatronics*

1. Introduction

Intelligent and Integrated Automation became one of the most studied areas inside the field of General Automation (mainly in areas such as manufacturing, agriculture, transport, medical care, domotics, military, space, etc.) because one of its goals is to make the design-product process with a high degree of integration and automation during all of its lifetime cycle.

Automated reasoning techniques can increase the productivity and the flexibility of design-product cycle operation. A great problem in automated reasoning is to design systems that can find automatically a set of actions (or a plan) that allow an agent to change the environment (called world) from a initial state to a desired state (goal state).

This plan could be delivered, for example, to a manufacturing system, a robot, or any kind of actuator that, following the plan, execute the actions and make the world change to the desired goal state. In Artificial Intelligence terminology the plan executor is called generically agent.

Therefore an intelligent manufacturing system, a robot or some other form of actuator with a control structure that can execute an action plan, collect information about the environment and use automated reasoning techniques about its perception and about what action to be done, can be called intelligent agent (Maes, 1991)(Russell, 1995).

Section 2 presents the planning problem complexity in real world applications and some systems already developed partially solving the problem. In Section 3 the ALWAYS$_{TRX}$ method is fully described, including definitions, the planning and execution modules with its algorithms and the implementation of the system with RETE algorithm adaptation. Finally Section 4 shows the final considerations and bibliographic references.

2. Problem complexity and related work

The first automatic planning system was created in the end of sixties and at that time many suppositions were made to decrease the complexity of the problem. One of those suppositions was the belief that the agent is the only element that can change the world. Another supposition was considering that the world changes are totally deterministic.

At that time the planning problem was to create techniques and heuristics to search in a tree of possible plans to achieve a desired goal. The complexity of this problem was analyzed years later and considered NP-Complete and NP-Hard (Noreils, 1995)(Erol, 1995).

Those suppositions early presented, together with some others, allowed the development of planning systems that create a action plan and when this plan is executed by an agent, the world is changed from its initial state to a goal state.

However, mainly due to the two suppositions cited, these planning systems were not adequate to applications in complex environments due to uncertainty and dynamics inherent to the real world. In many domains, like manufacturing systems, mobile robots, software agents, inspection, maintenance, surveillance, etc., the world state is changed by multiple agents and if the planner consider that it and only it can change the world, the resulting action plan will not represent the reality and the execution of this plan will be incompatible with the world where the agent is inserted. The agent is not the only element that can make changes in the world and it must use its perception resources to adequate the plan to the eventual contingencies during the plan execution.

Beyond the dynamics and unpredictability of the environment, the agent owns sensors and actuators that do not work in an ideal way and the time available for decisions is very limited. The knowledge base of the agent is incomplete and the planning systems must consider these aspects to make possible agent interaction in unstructured environments.

Early work integrating planning and execution like STRIPS/PLANEX (Fikes, 1971a)(Fikes, 1971b), Universal Plans (Schoppers, 1987), IPEM (Allen, 1990), BUMP (Olawsky, 1990), XII (Golden, 1994), SIPE-2/PRS-CL (Wilkins, 1988 (Georgeff, 1988) e SAGE (Knoblock, 1995), as well early works in planning with incomplete information like UCPOP (Penberthy, 1992), BURIDAN and C-BURIDAN (Kushmeric, 1995), among others, had been collaborated to the development of planning systems but partially solve the problem of plan in dynamic and unstructured environments because they do not consider, altogether, the aspects early presented. More detailed discussions can be seen in (Ash and Dabija, 2000).

Researches in planning systems that handle external events have been developed by our planning group since the eighties. Among the systems developed are: the PETRUS system (Rillo, 1988), EXTEPS (Rillo, 1992), PBE (Lopes, 1998) and finally, the ALWAYS$_{TRX}$, in its early research stage being presented in this paper. The ALWAYS$_{TRX}$ method is an effort to create systems that can effectively plan in complex, dynamic and unstructured real world domains.

3. ALWAYS$_{TRX}$ method

The ALWAYS$_{TRX}$ method (ALWAYS Thinking, Reacting and eXecuting) was designed to integrate the agent's planning task and execution task, including the external events treatment inside the planning phase, to give to the agent reactivity capabilities to the dynamic world changes in which it is inserted.

The integration approach requires that planning decisions followed by execution must be based in a common knowledge representation.

With ALWAYS$_{TRX}$ the planning system integration with the plan execution system considers that planning must be done concurrently with plan execution (the plan is made "on the fly"). The plan being created by the planner is represented by a tree where the nodes mean activities and events. Each path in the tree represents a sequence of activities with events that may occur. One path must be followed by the execution module. While the planner creates new paths, the execution module chooses one of these paths to execute, based on the actual occurrence of the external events associated with the chosen activities.

The planning system uses the information about which external events actually happened, obtained from the environment through the execution module, to adequate the paths to the real world state. As the execution module executes one step of the plan, the paths of the search tree not associated to the path that contains this step are excluded from the tree. One plan step means to observe the occurrence of an external event and initiate the execution of the correspondent activity. When a event occurs, the planner keep expanding paths below this event in the tree and exclude the paths that do not go through this event.

The ALWAYS$_{TRX}$ is domain independent and can be used in many engineering and automation applications such as intelligent manufacturing.

Figure 1 presents how the method is inserted in the context "see, think, act" and its internal architecture that consist of two modules running concurrently: the planning module and the execution module. Both modules constantly communicate through a common knowledge base.

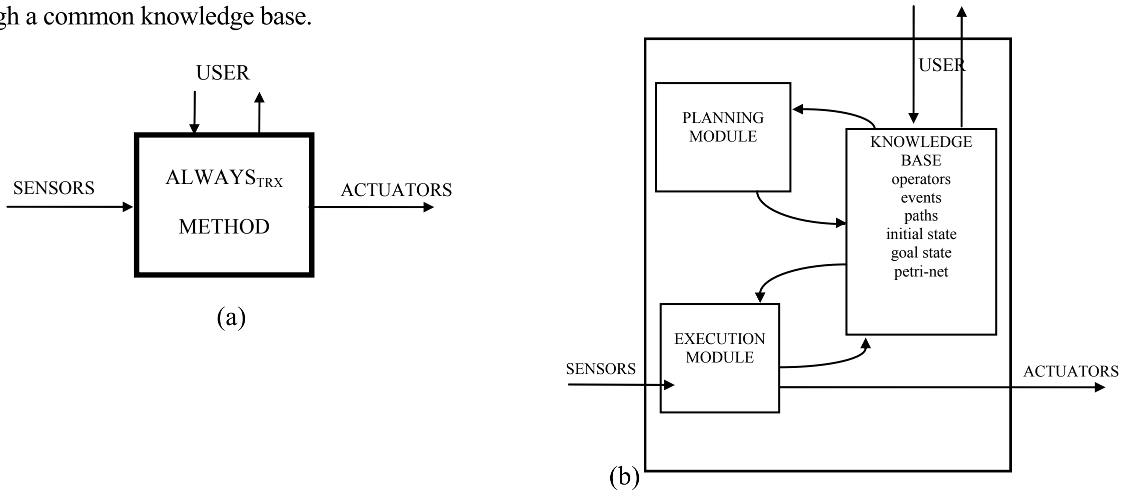


Figure 1. (a) ALWAYS$_{TRX}$ method in the context "see, think, act"; (b) ALWAYS$_{TRX}$ method's internal architecture.

As the planning module generates the plan, the paths are stored in the knowledge base. The execution module reads the knowledge base and follows the plan to be executed. As the execution module works, the activity being executed and all the information about which external events are happening are stored in the knowledge base. The information stored by execution module in the knowledge base is used by the planning module in order to keep synthesizing the plan paths with updated world information.

3.1. Concepts and Definitions

The definition of operator, event and world state follow the classic planning nomenclature (STRIPS) (Allen, 1990) with the addition of knowledge representation and the needed elements to make possible the treatment of external events as well the integration of the planning system with the execution control system. Below, some definitions used by ALWAYS$_{TRX}$ are presented.

An operator is defined as:

A generic activity with parameters (not instantiated), that represents one class of activities to be developed by the agent.

The instantiation of the operator's parameters represents a specific activity. For example, the operator *move(<u>object</u>, <u>from-place</u>, <u>to-place</u>)* represents a generic activity in which a agent moves any <u>*object*</u> from a <u>*from-place*</u> to a <u>*to-place*</u>. The parameters <u>*object*</u>, <u>*from-place*</u> and <u>*to-place*</u> are not instantiated variables. When variables are instantiated with, for example, ***green-box***, ***position12*** and ***position27***, respectively, the operator become an activity given by ***move(green-box, position12, position27)***.

One activity may be made of none, one or more actions, increasing the STRIP operator coverage that considers an instantiated operator as a unique action. There are cases that the agent must not execute any action and stay in a wait state until an event occurs. This case is considered in the ALWAYS$_{TRX}$ operator.

The actions to be executed by ALWAYS$_{TRX}$ system are, by definition, of two kinds: impulsive actions and continuous actions. The impulsive actions are executed instantly by the agent and finish. For example ***increment(variableN)***. The continuous actions represent tasks of repetitive or continuous cycles, as for example, ***follow(corridor2)***. ALWAYS$_{TRX}$ considers and handle these two kinds of actions. In the case of impulsive actions, the world state will have facts that indicate the consequences of the end of actions. With continuous actions the world state will have statements that the action(s) is(are) in execution.

The ALWAYS$_{TRX}$ operators are defined by the 4-tuple:

- operator name and parameters;
- pre-condition list;
- delete list;
- add list;

The pre-condition list contains facts and activities that must be satisfied in the world state, enabling the application of the operator. The delete list contains facts that will not be true after the beginning of execution of the activity. The add list contains the facts that will be true after the beginning of execution of the activity.

The ALWAYS$_{TRX}$ events are defined by the 5-tuple:

- -event name;
- -pre-condition list;
- -delete list;
- -add list;
- -class;

The pre-conditions delete and add lists are identical to ALWAYS$_{TRX}$ operators. The class is related to the possibility of event occurrence and is used by the planner to associate weights to the occurrence of paths.

The world state in ALWAYS$_{TRX}$ is defined by a set of facts, that can be predicates (for example ***Robot2(position1)***) and/or running activities (for example ***move(green-box, position12, position27)***). Each world state is associated with a weight, calculated from sensory information, that indicates the possibility of the state become true, based on information from the generated plan until that state.

3.2. Planning Algorithm

An example of the ALWAYS$_{TRX}$ method initially without the external events weights will be presented in this section. The user defines the operators and the events with all attributes as well the initial world state and the goal state and feed the system. After the data input, the user starts the system. The execution module still cannot do anything because there is no path to follow. The planner initiates its work. The graphic representation of the generated plans with its paths follows PBE (Lopes98) using Petri-nets and will be called here planning tree: the places represent a description of valid facts (world state) and the running activity. The running activity means none, one or more actions executing. The transitions represent the internal and external events that may occur and change the world when an activity is executing. Figure 2 presents a plan being generated (synthesized). From the initial world state description, all the activities (instantiated operators) with its pre-condition lists satisfied are selected. From the start event after the initial state will be *n* links corresponding to the *n* selected activities (A_i, A_j, ..., A_n Fig. 2).

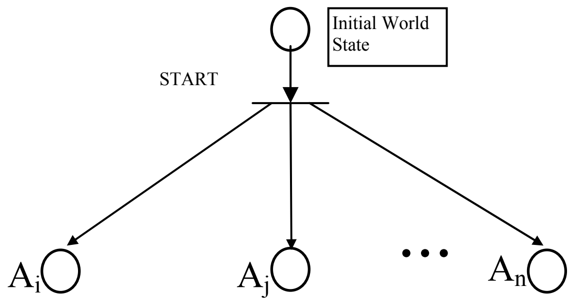


Figure 2. Selection of *n* possible activities (A_i, A_j, ..., A_n) from the initial world state, which pre-condition list satisfied.

Through an evaluation heuristic, one of the *n* activities is chosen as a first activity of the plant. The heuristic chooses the activity that best change the world towards the goal state. Next, the new world state is calculated removing the facts described in the selected activity delete-list and adding the facts described in the selected activity add-list.

The planner then finds the relevant events that may occur after the selected activity. The relevant events are all the events with its pre-condition lists satisfied (true in the calculated state). In the planning tree, below the selected activity, are *m* transitions corresponding to the *m* events that may occur after that activity (Fig. 3). The initial world state changed by activity A_j is called State A_j.

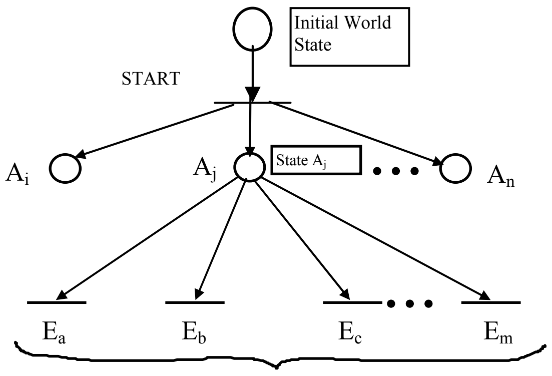


Figure 3 – Planning tree expansion with the relevant events to the State A_j, as a consequence from the activity A_j selected by the evaluation heuristic.

The next job of the planner is to compute for each relevant event the world state change after its occurrence, using the respective event's add-list and delete-list. Each transition in the planning tree correspond to an event and have a world state that corresponds to the application of the facts from the add-list and delete-list (Fig. 4).

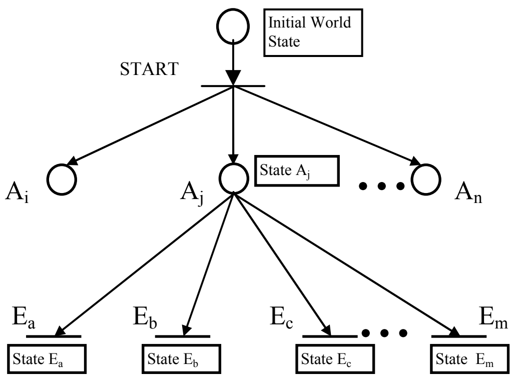


Figure 4 – New world states generated by the occurrence of the *m* events (E_a, E_b, E_c, ..., E_m).

At this point in the planning tree, each event has a world state associated. The planner repeat, for each event, the same procedure done after the START event: verifies the activities pre-condition lists (instantiating the operators) and select those with the pre-condition list satisfied (true) in the respective event state. Below each transition will be placed *r* links, corresponding to the *r* selected activities (Fig. 5).

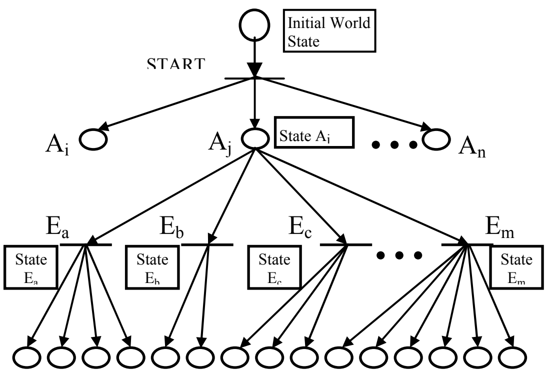
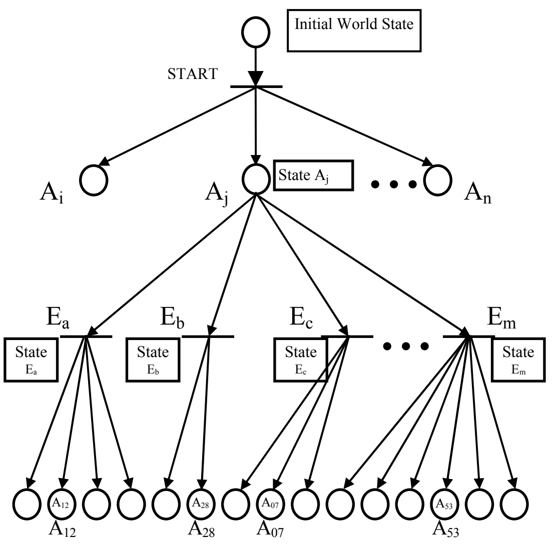


Figure 5 – Planning tree expansion with the relevant activities for each event.

The same evaluation heuristic early cited is used repeatedly for each transition, choosing one of the *r* activities as the best change to the world state towards to the goal state. The planning algorithm repeat the steps early described with the new data generated.

Below each one of the *m* transitions there will be an activity that was chosen to be the probable second path activity. It is important to note that now there are *m* paths with two activities, where the first activity is the same for all paths because it was the father node that generated the *m* transitions. Figure 6 show the selected activities and the generated paths.



m paths:	1.	A_j -> E_a -> A_{12}
	2.	A_j -> E_b -> A_{28}
	3.	A_j -> E_c -> A_{07}
	m.	A_j -> E_m -> A_{53}

Figure 6 – Selection of activities by the evaluation heuristic and m paths generated.

For each one of the *m* activities selected as a sequence of each one of the *m* transitions, the world state is computed using the add and delete lists and after that the pre-condition lists of all events are checked to the selection of the relevant event for each activity.

3.3. Execution Algorithm

Continuing the sequence explained in section 3.2, the execution module now have paths created by the planning module and can proceed starting its activities concurrently with the planning module. The execution module does not work with all planning tree and just uses the paths created by the planning module. In this example, Fig. 7 shows the resulting Petri-net. The START transition takes the system to state A_j, as a consequence of the A_j activity. The execution module register in the knowledge base that it began the execution of the activity A_j (put a mark on the place) and waits one of the *m* events that can happen after activity A_j, as described in the Petri-net of Fig. 7. With the execution of activity A_j, the other activities A_i, ..., A_n can not be anymore *backtracking points* and all the links in the tree below them will not be considered.

Suppose that event E_b happen. The execution module observing the occurrence of E_b, informs the knowledge base and verifies in the Petri-net the next activity to be executed, starting it immediately registering the fact in the knowledge base.

Each step of the execution module is registered in the knowledge base so that the planning module can simplify the planning tree, eliminating the links of the paths that do not contain any event or activity already executed by the execution module. The planner also updates the weights of occurrence of the plans.

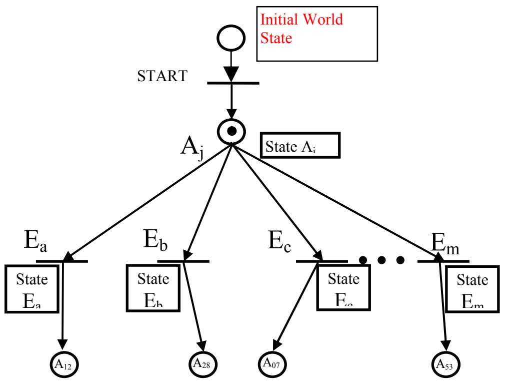


Figure 7 – Petri net viewed by the execution module. Mark on activity A_j to indicate "in execution".

The planning module, running concurrently with the execution module, work with a search tree of size much lower than a classic planner search tree. In each iteration, the expansion algorithm updates the tree changing it with new information from the execution module, until the goal state can be reached.

3.4. External Events Ocurrence Weights

The ALWAYS$_{TRX}$ planning module associates weights to the occurrence of the possible relevant external events that may come to happen in a given world state, based but not the same the probabilistic reasoning with Bayesian networks (Pearl, 1988)(Abranson, 1990). Considering the weights of many queued events with the activities, is possible to compute possibilities of reach a given world state. With this information, the planning algorithm can expand the planning tree more deeply in those paths with greater weight and expand superficially in paths with lower weights. This process also decreases the computational effort because not all the paths are fully expanded.

3.5. Implementation

The ALWAYS$_{TRX}$ method was implemented in C language for Windows and UNIX platforms with an user interface directed to collect information about operators, events, initial world state and goal state. Information during the agent execution can be viewed in the knowledge base.

As early noted, the complexity of a planning system handling external events integrated with agent execution demand optimization techniques in the computer implementation otherwise it could not be used in real world applications.

One fundamental technique used in the implementation of ALWAYS$_{TRX}$ method was the adaptation of RETE algorithm (Forgy82)(Russel95) to solve the problem of variable instantiation in the pre-condition lists of the operators and events. One RETE like network is created before the system START and is used by the planning module.

The information about operators and events given by the user are checked for inconsistencies and, if everything is ok, are compiled to a RETE-like network.

The user then STARTs the system establishing the knowledge base and the two concurrent processes corresponding to the planning module and the execution module.

4. Final Considerations

A new planning method that handles external events in the planning phase and is integrated to the execution control of intelligent manufacturing systems, called ALWAYS$_{TRX}$, was presented.

The planning module works concurrently with the execution module providing to the agent (e.g. manufacturing system) reactive and planning skills considering external events. The planning module considers the uncertainty in the occurrence of external events associating weights to discover which world state has greater possibility of happen. The ALWAYS$_{TRX}$ method works with weights to update the planning tree to reach a given state. Additional studies must be carried out to find out ways of associate weights to the external events, as well techniques to observe the behavior of the occurrence of events and ways of update the weight values considered initially.

The paths generated a little bit earlier by the planner to the execution module are represented by a Petri-net and the execution module reads the Petri-net and executes as a Petri-net player.

At this point the generated Petri-net works for all possible states only one mark and therefore in this particular case the Petri-net is a State Diagram. Further studies must be made to use multiples marks, making explicit the execution parallelism and the concept of partial state.

5. References

Abramson, B.; Ng, K., 1990, Uncertainty Management in Expert Systems. **Ieee Expert**, pp.29-48, Apr.

Allen, J.; Hendler, J.; Tate, A., 1990, **Readings in Planning**. San Mateo, California, Morgan Kaufmann.

Ash, D.J.; Dabija, V.G., 2000, **Planning for Real Time Event Response Management,** Prentice Hall PTR.

Erol, K.; Nau, D.; Subrahmanian, V.S., 1995, Complexity, Decidability and Undecidability Results For Domain Independent Planning. **Artificial Intelligence**, n.76, pp.75-88.

Fikes, R.E.; Nilsson, N.J., 1971a, Strips: A New Approach to the Application of Theorem Proving to Problem Solving. **Artificial Intelligence**, n.2, pp.189-208.

Fikes, R.E., 1971b, Monitored Execution of Robot Plans Produced by Strips. **Conf. IFIP**, Ljubljana, Yu, Aug. **Proceedings**.

Fikes, R.E.; Nilsson, N.J., 1993, Strips: A Retrospective. **Artificial Intelligence**, n.59, pp.227-232.

Forgy, C.L., 1982, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. **Artificial Intelligence**, v.19, n.1, pp.17-37.

Georgeff, M.P.Et Al., 1988, Reasoning About Plans and Actions. **Exploring Artificial Intelligence**, Morgan Kaufmann, AAAI, p.5, pp.173-196.

Golden, K.; Etzioni, W.; Weld, D., 1994, Omnipotence without Omniscience: Efficient Sensor Management for Planning. In: **National Conference On Artificial Intelligence**, 12., AAAI Press. **Proceedings**. pp.1048-1054.

Knoblock, C., 1995, Planning, Executing, Sensing and Replanning for Information Gathering. In: **IJCAI**, 14., Montreal. **Proceedings**. pp.1686-1693.

Kushmeric, N.; Hanks, S.; Wold, D. An Algorithm for Probabilistic Planning. **Artificial Intelligence**, v.1-2, n.76, pp.239-286.

Lopes, C., 1998, **Planejamento Baseado em Expectativas.** São Paulo, 1998, Tese de Doutorado, Departamento de Engenharia de Eletricidade, Epusp.

Maes, P., 1991, **Designing Autonomous Agents.** Mit/Elsevier.

Mcdermott, D., 1994, The Current State of AI Planning Research. In: International Conference on Industrial and Engineering Applications of AI and Expert Systems. **Proceedings**.

Noreils, F.R.; Chatila, R.G., 1995, Plan Execution Monitoring and Control Architecture for Mobile Robots. **IEEE Transactions on Robotics and Automation**, v.11, n.2, Apr.

Olawsky, D.; Gini, M., 1990, Deferred Planning and Sensor Use. In: Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, Ca, Darpa. **Proceedings**. pp.166-174.

Pearl, J., 1988, **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.** San Mateo, California, Morgan Kauffmann.

Pemberthy, J.; Weld, D., 1992, Ucpop: A Sound, Complete, Partial Order Planner for ADL. In: International Conference on Principles of Knowledge Representation and Reasoning, 3.. **Proceedings**. pp.103-114.

Rillo, M., 1988, **Aplicações de Redes de Petri em Sistemas de Manufatura.** São Paulo, Tese de Doutorado, Departamento de Engenharia de Eletricidade, EPUSP.

Rillo, M., 1992, Expectation-Based Temporal Projection System. In: Annual Conference on AI, Simulation and Planning in High Autonomy Systems, 3., Perth, Australia, July. **Proceedings**. pp.276-281.

Russel, S.; Norvig, P., 1995, **Artificial Intelligence: A Modern Approach.** Prentice Hall.

Schoppers, M.J., 1987, Universal Plans for Reactive Robots in Unpredictable Environments. In: IJCAI, 10., Milan. **Proceedings**. pp.1039-1046.

Wilkins, D.E., 1988, **Practical Planning: Extending the Classical AI Planning Paradigm.** Morgan Kauffmann.

5. Responsibility notice

The authors are the only responsible for the printed material included in this paper.